Learning Source Descriptions for Data Integration

AnHai Doan, Pedro Domingos, Alon Levy
Department of Computer Science and Engineering
University of Washington, Seattle, WA 98195
{anhai, pedrod, alon}@cs.washington.edu

ABSTRACT

To build a data-integration system, the application designer must specify a mediated schema and supply the descriptions of data sources. A source description contains a source schema that describes the content of the source, and a mapping between the corresponding elements of the source schema and the mediated schema. Manually constructing these mappings is both labor-intensive and error-prone, and has proven to be a major bottleneck in deploying large-scale data integration systems in practice. In this paper we report on our initial work toward automatically learning mappings between source schemas and the mediated schema. Specifically, we investigate finding one-to-one mappings for the leaf elements of source schemas. We describe LSD, a system that automatically finds such mappings. LSD consults a set of learner modules – where each module looks at the problem from a different perspective, then combines the predictions of the modules using a meta-learner. We report on experimental results of applying LSD to five sources in the real-estate domain.

1. INTRODUCTION

The rapid growth of information available online has spurred numerous research activities on developing data integration systems (e.g., [4, 8, 5, 7, 6]). A data integration system provides a uniform interface to a multitude of data sources: given a user query formulated in this interface, the system accesses and combines data from the sources to produce answers to the query.

To build a data integration system, the application designer begins by developing a *mediated schema* that captures the relevant aspects of the domain of interest. Along with the mediated schema, the application designer needs to supply descriptions of the data sources. A *source description* specifies the semantic mapping between the schema of the data source and the mediated schema.

Constructing source descriptions is one of the key bottlenecks in creating data integration applications that query a large number of sources. Currently, source descriptions are created manually in a very labor-intensive and error-prone process. As data sharing on the WWW becomes pervasive with the adoption of XML, the problem of reconciling schemas (DTDs, XML schemas) is only exacerbated.

In this paper we report the first results of our work on using machine learning to (semi-)automatically compute semantic mappings between schemas. The idea underlying our approach is that after a set of data sources have been manually mapped to a mediated schema, the system should be able to glean significant information from these mappings and to successfully propose mappings for subsequent data sources.

Example 1: Consider a data integration system that helps users find houses on the real-estate market. A mediated schema for this domain may contain elements house_address, price, and contact_phone, listing the address of a house, the price, and the phone of the contact person, respectively (see Figure 1).

Suppose we have a source *realestate.com*, for which we provide the source description manually. Specifically, suppose the source contains the elements house_location, listed_price, and agent_phone (Figure 1), and the mapping specifies that these elements match the elements house_address, price, and contact_phone of the mediated schema, respectively.

There are several things that a machine learning program can glean from such a mapping. First, if it looks at the data in the source, it now has many examples of home addresses, home prices and phone numbers, and it can therefore create recognizers for these elements. Second, the system can learn by looking at the names of the elements. For example, knowing that source element agent_phone matches contact_phone, it may hypothesize that the word "agent" (as well as "phone") in an element name is indicative of that element being contact_phone. The system can also learn from the properties of the data. For example, small numbers tend to indicate numbers of rooms, not prices of houses. As another example, when the phone numbers of a given element have significant commonalities, the phone numbers are more likely to be the office phones of employees, rather than home phones. Finally, the program can learn from the proximity of elements. For example, in the real-estate domain it often happens that a long text field at the beginning of the house entry is the house description, or that agent phones tend to appear next to the name of the real-estate agency.□

In general, there are many different types of information that a learner can exploit, such as names, formats, word frequencies, positions, and characteristics of value distribution. Clearly, no single learner will be able to exploit effectively all such types of information. Hence, our work takes a multi-strategy learning approach. We apply a set of learn-

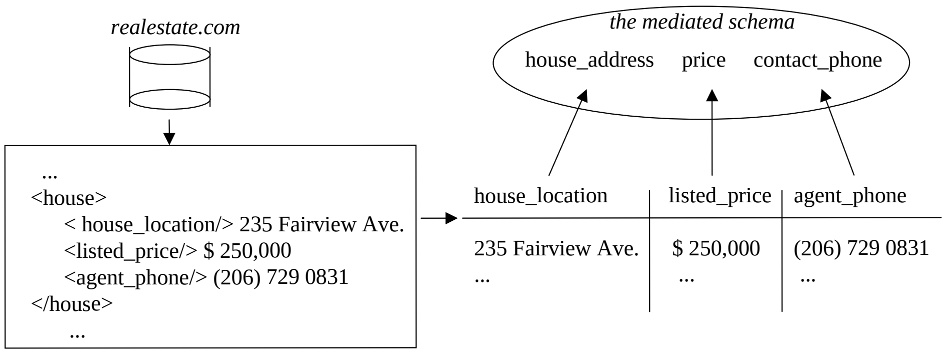

Figure 1: Source *realestate.com* returns data that is in its local schema, which then needs to be mapped to the mediated schema of the data integration system.

ers, each of which learns well certain kinds of patterns, and then the predictions of the learners are combined using a meta-learner. In addition to providing accuracy superior to any single learner, this technique has the advantage of being extensible when new learners are developed.

We describe the LSD (Learning Source Descriptions) system that we built for testing this approach, and our initial experimental results. The results show that with the current set of three learners, we already obtain predictive accuracy of 62-75% prediction in a fairly complex domain of real-estate data sources. Our work currently focuses on finding one-to-one mappings for the leaf elements of source schemas.

Learning Source Descriptions versus Wrapper Learning: We emphasize that the learning problem we consider here is different from the problem of learning wrappers. *Wrappers* are programs that convert data coming from a source (say, in HTML format) into some structured representation that a data integration system can manipulate (e.g., XML). In wrapper learning, the focus is on extracting the structure from the HTML pages. By contrast, here the focus is on finding the *semantic* mapping between the tags/attributes in the data source and those in the mediated schema. Hence, throughout the discussion we assume that the data in the source is already given to us in XML.

2. THE SCHEMA-MATCHING PROBLEM

Schema Definition: We model a schema with a tree the nodes of which are XML tag names. Figure 2 shows a fragment of a mediated schema G and a fragment of a source schema M. We also refer to the nodes of the schema tree as *schema elements*. Each schema element has a *name* and *values* (also called *instances*). In a schema tree, element B being a child of element A simply means that an instance of A *may* contain an instance of B. For example, in G the name of the schema element corresponding to the address of a house is house_address, and its instances are text strings specifying the address, such as "123 Fairview Ave., Seattle, WA 98135" or "4028 13th Str.". The former string contains an instance of element city ("Seattle"), while the latter does not. As such, our schema language can be considered a simplified version of XML DTD, where parent = child1?child2?...childn?.

The Schema-Matching Problem: Given a mediated-schema tree G and a source-schema tree M, both expressed in the above schema language, in general the schema matching problem is to find some mappings between the two schemas. The simplest type of mapping is 1-1 mapping between a node in the source-schema tree and a node in the mediated-schema tree, such as the mappings shown in Figure 1 and mentioned in Example 1. More complex types of mapping include mappings from a node in a tree into several nodes in the other tree (e.g., num_bathrooms in one schema is the sum of num_full_bathrooms and num_half_bathrooms in the other), and mappings between a node in a schema and the values of another node in the other schema (e.g., handicap_equipped with values "yes/no" in one schema maps into the value "handicap equipment available" of amenities in another schema).

As the first step, we focus on finding all 1-1 mappings between the nodes (elements) of the two schema trees. Specifically, in this paper, we limit our investigations to finding 1-1 mappings for the leaf elements of source schemas. Matching source-schema elements at higher levels requires developing learning methods that deal with structures, a topic we are currently exploring.

3. THE LSD APPROACH

We now explain in detail how our machine-learning approach works. In our discussion we shall use *labels* to refer to mediated-schema elements. We refer to the process of matching a source-schema element as *classifying* the element and assigning to it a proper label.

The Learning Phase: Suppose we start with the real-estate mediated schema G shown in Figure 3.a. We create training data for the learning process by manually matching schema elements of source P with the elements of the mediated schema. Figure 3.c shows that source element *a* has been matched with mediated-schema element *A*, and *b* matched with *B*.

Next, we extract a set of house objects from source P. Figure 3.d shows two extracted house objects in XML. We train the learners using the extracted data. Each learner tries to learn the mediated-schema elements A and B, so that when presented with an element from a new source, the learner can predict whether it is A, B, or neither.

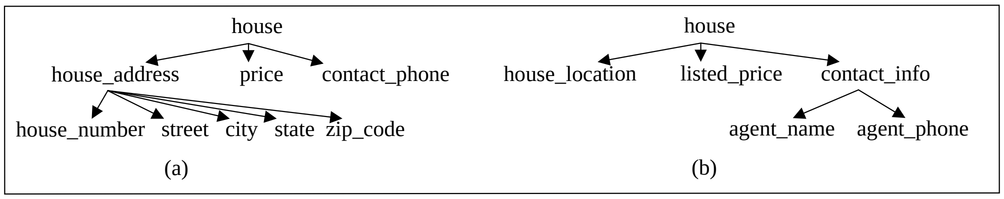

Figure 2: Schema fragments for the real-estate domain: (a) mediated schema, (b) source schema.

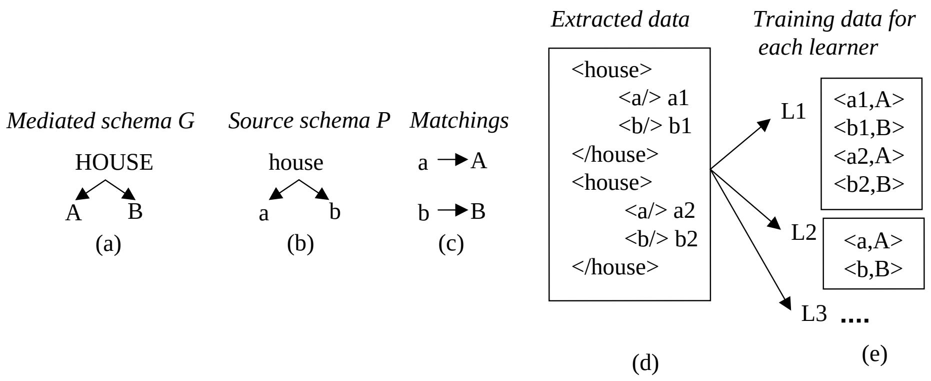

Figure 3: The learning phase for LSD.

Even though the goal of all learners is the same, each of them learns from a different type of information available in the extracted data. So each learner processes the extracted data into a different set of training examples. For example, learner L_1 may deal only with *instances*, so it extracts the instances a_1, a_2, b_1, b_2 and forms the training examples shown in Figure 3.e. Example $\langle a_1, A\rangle$ means that if a source element contains an instance a_1, then that source element matches A. L_1 can form this example because a_1 is an instance of source element a, and we have told it that a matches A.

As another example, suppose learner L_2 deals only with *element names*. Then it forms two training examples as shown in Figure 3.e. Example $\langle a, A\rangle$ means that if element name is a, then it matches A.

The Classification Phase: Once the learners have been trained, we are ready to perform schema matching on new sources. Suppose we would like to classify schema element m of source Q (Figure 4.a). We begin by extracting a set of house objects from Q. Figure 4.b shows three extracted house objects. Next, we consider each house object in turn. Take the first house object in Figure 4.b. From this house object we extract and send each learner appropriate information about schema element m (Figure 4.c). Since learner L_1 can only deal with instances, we send it instance m_1; since learner L_2 can only deal with names, we send it the name m, and so on.

Each learner will return a prediction list $\{\langle A, s_1\rangle, \langle B, s_2\rangle\}$, which says that based on the data of the first house object, it predicts that schema element m matches A with confidence score s_1, and matches B_1 with score s_2. The higher the confidence score, the more certain the learner is in its prediction.

A meta learner then combines the predictions of all learners to form a final prediction (Figure 4.d). For example, the meta learner may predict A, which means that based only on data of the first house object, the learners, combined, think that m matches A.

We proceed in a similar manner for subsequent house objects. At the end of this process, we have obtained a prediction list for m that contain one prediction per house object (Figure 4.e). A prediction combiner then uses the list to predict a final matching result for element m. For example, the list shown in Figure 4.e is $\{A, A, B\}$. Based on this list, the prediction combiner may decide that m best matches A (Figure 4.f).

The Learners: In principle, any learner that can issue label predictions weighted by confidence score can be used. The current implementation of LSD has four modules: a nearest neighbor Whirl learner, a Naive Bayesian learner, a name matcher, and a county-name recognizer.

The Whirl Learner classifies an input instance based on the labels of its nearest neighbors in the training set [2]. It uses the TF/IDF similarity measure commonly employed in information retrieval. Whirl performs best on schema elements whose values are *verbose* and *textual*, such as house descriptions (free-text paragraphs), or *limited* but *uniquely indicative* of the type of the element, such as color (red, green, yellow, etc).

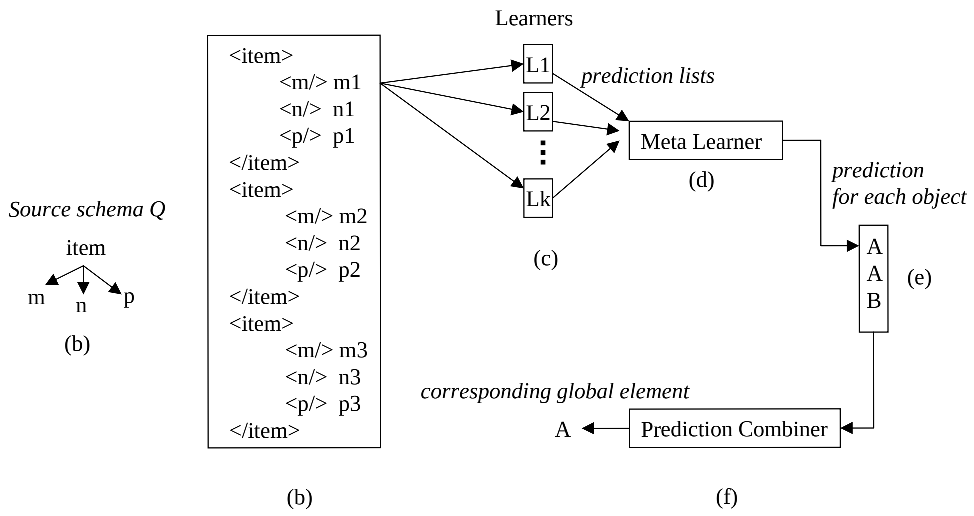

Figure 4: The classification phase for LSD.

The Naive Bayesian Learner exploits word frequencies [3], and works best when there are words that are strongly indicative of the correct label, by the virtue of their frequencies. For example, it works for house descriptions which frequently contain words such as "beautiful" and "fantastic" – words that seldom appear in other elements. It also works well when there are only weakly suggestive words, but many of them. It does not work well for short or numeric fields, such as color, zip code, or number of bathrooms.

The Name Matcher matches schema elements based on the similarity of their names, allowing synonyms. It also uses the TF/IDF similarity measure. This learner works well on unambiguous names (e.g., price or house_location), and fails on names that are either ambiguous (e.g., office to indicate office phone) or too general (e.g., item).

The County-Name Recognizer searches a database pulled from the Web to verify if an instance is a county name. This module illustrates how recognizers with a narrow and specific area of expertise can be incorporated into our system.

The Meta Learner: The meta learner combines the predictions of the base-level learners using a machine learning method called *stacking* [14, 13]. In LSD, the meta learner is a linear discriminant machine. It uses the training data to learn for each combination of label and base-level learner a weight that indicates the relative importance of that learner for that label. Details of this learning process can be found in [13].

Then given an input instance, for each label the meta learner computes the weighted sum of the confidence scores that the base-level learners give for that label. It assigns the label with the highest weighted sum to the input instance.

The Prediction Combiner: This module uses the following simple heuristic to assign label to a source-schema element Q: Let T be the set of instances of Q that have been labeled. Suppose the label associated with the highest number of instances in T is L_1, and the label with the next highest number of instances is L_2. If L_1 is the label of at least p% of instances in T, and $|L_1 - L_2| \geq q$, where p and q are prespecified thresholds, then assign label L_1 to Q. Otherwise, report failure to assign label to Q. This heuristic is similar to the heuristic used in [12] for the same purpose.

4. EXPERIMENTS

We have carried out preliminary experiments to evaluate the feasibility of our approach. We tested LSD on five real-estate sources that list houses for sale. Figure 5 shows the sources and their characteristics. These sources have a broad range of schema elements, from short ones such as num_bathrooms (numeric values) to very long ones such as house_description (free-text paragraphs). They contain elements of special formats, such as phone number and email, as well as elements whose successful classification requires knowledge beyond what is available in the schema and data. Finally, they also contain elements that do not have 1-1 matchings in the mediated schema. In short, these five real-estate sources present a challenging test domain for schema-matching algorithms.

We started by extracting 300 house objects from each source. Next we performed ten experiments, in each of which we picked three sources for training and two sources for testing. The system is trained on data from the training sources, and tested on data from the remaining sources.

The last two columns of Figure 5 show the average accuracy rate for each source. For example the numbers 24/31 in the first row means that on average the system correctly classified 24 out of 31 classifiable leaf elements of source *realestate.yahoo*. (An element is classifiable if it has an 1-1 matching in the mediated schema.) This corresponds to a 77% classification accuracy.

The results show that LSD performed quite well on the five sources, with accuracies ranging from low 60% to high 70%. It is important to emphasize that 100% accuracy is unlikely to be reached, simply because it is difficult even for humans

Sources	Coverage	# elem	# leaf elems	# class. elems	Min-max	Heavy textual	Numeric	Spe cial	Domain Know.	Avg. Accuracy	Per cent
realestate.yahoo	national	31	31	31	1 – 152	3	6	10	0	24/31	77%
homeseekers.com	national	33	31	31	1 – 138	2	5	8	0	20/31	64%
nkymls.com	Northern Kentucky	82	64	28	1 – 56	2	6	6	0	21/28	75%
texasproperties.com	Texas	56	52	42	1 – 110	2	10	14	4	26/42	62%
windermere.com	Northwest	39	35	35	1 – 87	3	4	8	1	22/35	63%

Figure 5: The characteristics and average classification accuracies of the five real-estate sources: *# elems:* number of source-schema elements; *# leaf elems:* number of leaf elements; *# class. elems:* number of leaf elements that are classifiable, i.e., having matching mediated-schema elements; *Min-max:* the minimal and maximal length of element instances, measured in words; *Heavy textual:* number of classifiable elements that are heavily textual; *Numeric:* number of classifiable elements that are numeric; *Special:* number of classifiable elements that have special format (such as phone number); *Domain Know.:* number of classifiable elements that require domain knowledge to be successfully classified; *Avg. Accuracy:* number of correctly classified leaf elements/number of classifiable leaf elements; *Percent:* average accuracy, in terms of percentage.

to reach that accuracy level. So the questions we consider now are (a) how much higher accuracy we can obtain, and (b) how to achieve that degree of accuracy.

To answer these questions, we are currently identifying reasons that prevent LSD from correctly matching the remaining 30-40% of the elements, and considering extensions to help obtain higher accuracy. A major reason that causes LSD to fail on some elements is unfamiliarity with that element type. For example, LSD could not match element suburb because it has never seen the word "suburb" in the name of an element before, nor did it recognize the instances of this element to be the names of the suburbs. This problem could be handled by adding more recognizers. LSD also did not do very well where there are only subtle or subjective distinctions, or no clear boundary among elements. For example, it failed to distinguish between lot description and house description (both free-text paragraphs). A quick fix to this is to concatenate all the instances of an element together to form a mega document, then classify mega documents instead of individual instances. We believe that concatenating instances may amplify the subtle distinction among elements to the extent that the current learners can distinguish them.

5. RELATED WORK

Work on (partially) automating the schema matching process can be classified into rule-based and learner-based approaches. Works in the rule-based approach include [10, 11, 1]. The Transcm system [10] performs matching based on the *name* and *structure* of schema elements. Both schemas M and G are represented with tree structures with labeled nodes, a node X in M matches a node Y in G if X label matches Y label (allowing synonymous labels) and/or certain child nodes of X also match certain child nodes of Y. Therefore, in the case node X of M is a leaf element, Transcm's work amounts to using only the name matcher to match X. The Artemis system [1] uses names, structures, as well as domain types of schema elements to match schemas. In general, existing rule-based systems utilize only information inherent in the *schemas*, whereas our approach exploits both schemas and *data* (i.e., values of schema elements), to perform matching.

Works in the learner-based approach include [9, 12]. The Semint system [9] uses a neural-net learner. The ILA system [12] matches schemas based on comparing objects that it knows to be the same in both the source and the mediated schema. Both ILA and Semint employ a single type of learner and therefore have limited applicability. For example, ILA works only if sources share some objects and these objects can be identified. Object identification cannot be done effectively in the real-estate domain, where many brokerage firms go to great length to conceal house identity (by giving only a partial house address) to avoid being left out of the process. Another example is that the neural net of Semint cannot deal effectively with textual elements. However, we note that learners that have been studied by works in this approach can be easily incorporated into our system. In fact, in the next version of LSD we intend to add the object-based learner of ILA.

6. CONCLUSIONS AND FUTURE WORK

We aim to build data integration systems that can explore the Web, find relevant sources, add them to the systems, and automatically learn their descriptions. Towards this end, we have built LSD, a prototype system that uses machine learning to automatically match source schemas with the mediated schema. LSD employs a diverse range of learners each of which draws knowledge from a different source. As such, LSD can easily incorporate the previous approaches, and provides a broader range of applicability. We applied LSD to five sources in the real-estate domain. The experimental results demonstrate the feasibility and promise of our approach.

In this paper we have focused on finding 1-1 mappings for the leaf elements of source schemas. In the near future we would like to explore a range of learning issues. We plan to continue working on improving LSD's classification accuracy. We want to add more learners to LSD and develop new methods to combine the learners effectively. We are also interested in learning recognizers with narrow areas of expertise, which can be applied to many domains. We plan to investigate the impact of such recognizers on classification accuracy.

Next, we plan to work on incorporating domain knowledge such as semantic integrity constraints into the learning process. Real-world domains usually contain considerable amount of nested structure among elements. It is very important that we develop techniques to exploit such structure, so that we can effectively match elements higher up in source-schema trees. Finally, we shall also look at learning matchings that are more complex than 1-1 matchings, and consider extensions to the mediated schema. In particular, mediated-schema elements usually form a natural hierarchy of concepts. We would like to investigate how to exploit such hierarchies to improve learning.

7. ACKNOWLEDGMENTS

This work is supported in part by ARPA/Rome Labs Grant F30602–95–1–0024, NSF grant IRI–9523649, NSF grant IIS-9978567, and a Sloan fellowship.

8. REFERENCES

[1] S. Castano and V. D. Antonellis. A schema analysis and reconciliation tool environment for heterogeneous databases. In *Proc. of the Int. Database Engineering and Applications Symposium (IDEAS-99)*, pages 53–62.

[2] W. W. Cohen and H. Hirsh. Joins that generalize: Text classification using whirl. In *Proc. of the Fourth Int. Conf. on Knowledge Discovery and Data Mining (KDD-98)*, 1998.

[3] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simple Bayesian classifier. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 105–112, Bari, Italy, 1996. Morgan Kaufmann.

[4] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[5] Z. G. Ives, D. Florescu, M. A. Friedman, A. Y. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, pages 299–310. ACM Press, 1999.

[6] C. Knoblock, S. Minton, J. Ambite, N. Ashish, P. Modi, I. Muslea, A. Philpot, and S. Tejada. Modeling web sources for information integration. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 1998.

[7] E. Lambrecht, S. Kambhampati, and S. Gnanaprakasam. Optimizing recursive information gathering plans. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[8] A. Y. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 251–262, 1996.

[9] W.-S. Li and C. Clifton. Semantic integration in heterogeneous databases using neural networks. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. of the 20th Int. Conf. on Very Large Data Bases (VLDB-94)*, pages 1–12, 1994.

[10] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proc. of the 24th Int. Conf. on Very Large Data Bases (VLDB-98)*, pages 122–133, 24–27 Aug. 1998.

[11] L. Palopoli, D. Sacca, and D. Ursino. Semi-automatic, semantic discovery of properties from database schemes. In *Proc. of the Int. Database Engineering and Applications Symposium (IDEAS-98)*, pages 244–253.

[12] M. Perkowitz and O. Etzioni. Category translation: Learning to understand information on the Internet. In *Proc. of IJCAI-95*, pages 930–936, 1995.

[13] K. M. Ting and I. H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

[14] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.